Execution of Queueing Network Analysis Algorithms On a Data Flow Computer Architecture

Kenneth C. Sevcik
Peter J. Denning

August, 1985

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS TR 85.9

Research Institute for Advanced Computer Science

Execution of Queueing Network Analysis Algorithms On A Data Flow Computer Architecture

Kenneth C. Sevcik
Peter J. Denning

RIACS
NASA Ames Research Center

August, 1985

1. Introduction

Parallel computer architectures of many varieties have been developed over the past several years, and new types continue to emerge. It is important to investigate the extent to which such architectures allow specific computational tasks of recognized significance to be carried out more quickly than is possible on conventional sequential machines.

A recent study at RIACS [A85] investigated this issue with respect to six types of problems, all known for their massive computational requirements when treating "large" problems. The six areas were computational fluid dynamics, computational chemistry, galactic simulation, linear system solution, natural language processing, and queueing network analysis. The specific parallel architecture studied was data flow architecture [D84].

In this report, we focus on the area of queueing network analysis. We give a more detailed discussion of the investigation from which summary results were included in the report that described the entire data flow architecture evaluation study [A85].

2. Problem Overview

Queueing Network Models are tools for deriving performance estimates for computer systems and communications networks [DB78]. From inputs that describe *workload intensities* (the volumes of transactions or jobs or messages) and *service demands* (the average amounts of service at each system device, or *center*, required on average by a work unit), it is possible to calculate such performance measures as throughputs, average response times, utilizations, and average queue lengths. In *single-class* queueing network models, the assumption is made that all items of work are statistically identical. In *multiple-class* models, on the other hand, the workload intensity and the service demands of each distinguishable workload component are described individually.

In a single-class queueing network model with N customers and K devices, there are

$$L = \left(\begin{matrix} N+K-1 \\ N \end{matrix} \right) = \frac{(N+K-1)!}{N!(K-1)!}$$

distinct system states. (That is, there are L distinguishable permutations of N indistinguishable customers and $(K-1)$ device queue separators.) Since the placement of customers in various classes is independent, the corresponding expression for a multiple-class network is

$$L = \prod_{i=1}^{C} \binom{N_i + K - 1}{N_i}$$

where C is the number of classes, and N_i is the population of class i.

By the technique of *global balance* [LZGS84], the equilibrium state probabilities for such networks can be calculated by solving L simultaneous linear equations in which the unknowns are the equilibrium probabilities of the system states. Clearly, however, this is computationally feasible only for networks with very few classes, few devices, and few customers per class. With two classes, five devices, and ten customers per class, the number of simultaneous linear equations to be solved already exceeds one million.

Fortunately, an important subset of queueing network models (those that satisfy certain restrictions leading to the property of *separability* [LZGS84]) can be solved efficiently. Techniques are known for obtaining performance estimates directly, rather than by summing state probabilities over subsets of system states. For example, using *mean value analysis (MVA)* [RL80], the throughput and average response time of a single-class queueing network model with N customers and service demands, $D_1, D_2, \ldots, D_K$ at the K centers can be expressed recursively by:

$$\text{Throughput}: \quad X(N) = N / R(N)$$

$$\text{Mean Response Time}: \quad R(N) = \sum_{i=1}^{K} R_i(N) \tag{1}$$

$$\text{where} \quad R_i(N) = D_i \left[1 + \frac{R_i(N-1)}{R(N-1)} (N-1) \right]$$

$$\text{with} \quad R_i(1) = D_i$$

By a different technique, called *convolution* or *normalization constant analysis* [B73, RK75], the same performance measures can be expressed as

$$\text{Throughput}: \quad X(N) = G(N-1) / G(N)$$

$$\text{Mean Response Time}: \quad R(N) = \frac{N \; G(N)}{G(N-1)} \tag{2}$$

$$\text{where} \quad G(N) = g(N, K) \tag{3}$$

$$\text{and} \quad g(n,k) = g(n,k-1) + D_k \times g(n-1,k) \tag{3}$$

$$\text{with} \quad g(0,k) = 1 \quad k = 0,1,2, \cdots, K$$

$$\text{and} \quad g(n,0) = 0 \quad n = 1,2, \cdots, N$$

This recursive expression for $G(N)$ is a computationally efficient means of calculating the normalization constant that assures that the state probabilities sum

to one, and is defined by

$$G(N) = \sum_{\substack{\vec{n}=(n_1,n_2,\cdots,n_K) \\ s.t. \sum_i n_i = N}} \prod_{i=1}^{K} (D_i)^{n_i} \qquad (4)$$

Later in the paper, we will refer to both normalization constant analysis based on recursion (*NCR*), which uses equations (3) to compute $G(N)$, and normalization constant analysis based on direct computation (*NCD*) of $G(N)$ by equation (4).

Calculation of throughput and mean response time by either MVA or NCR requires no more than about $4KC \prod_i (N_i + 1)$ arithmetic operations. For the example network mentioned earlier with two classes, five centers and ten customers in each class, the solution requires about 5000 operations with either efficient technique as opposed to the millions or billions of operations that would be required by the global balance solution which involves the solution of a million simultaneous linear equations in a million unknowns (even taking full advantage of the sparseness of the coefficient matrix). The NCD approach would require around 20 million operations since the summation is over more than a million terms, and each term consists of 20 factors.

For very large queueing network models, exact solution even by mean value analysis or normalization constant analysis can become computationally very expensive. For example, for five classes, ten centers, and one hundred customers per class, the efficient exact solution techniques would require over one trillion arithmetic operations. Consequently, it is of interest to investigate the extent to which parallel computer architectures can increase the size of queueing network models for which it is feasible to obtain the exact solutions.

3. Abstract Algorithms

We will consider three methods of obtaining exact queueing network model solutions using a dataflow architecture. The first method (MVA) involves using the mean value analysis recursion (equations (1)), while the other two involve calculating the normalization constant either recursively (NCR using equations (3)) or directly (NCD using equations (4)), and then using equations (2) to obtain the performance measures. Note that the NCD method is the "most parallel" in the sense that all the terms of the summation are computed in parallel, and then the sum of all the terms is calculated. A common approach when developing algorithms for implementation on parallel architectures is to start with the statement of the problem that imposes the fewest restrictions on parallelism. We shall see that in the case of the problem under investigation here, such an approach does not lead to the best solution. Figures 1, 2, and 3 show the dataflow diagrams for the three algorithms respectively. Part A of each figure treats the single class case, and part B the multiple class case. In the diagrams, we assume that the fan-in to and fan-out from each operation is restricted to two.

4. Algorithm Performance

In order to obtain some performance estimates, we will assume here that the VAL compiler succeeds in mapping the data flow diagram onto the data flow machine processing elements in an optimal way, so that the arithmetic units are kept busy all the time. This assumption is quite optimistic and will bias our comparisons in favor of the data flow architecture. Tables 1 and 2 summarize the results of analyzing the various approaches for single class and multiple class models respectively. In each table, we indicate the space and sequential operations required for algorithm execution on conventional architectures. For data flow architectures, we give estimates of the number of operator nodes in the full data flow diagram, and the number of time steps required for data flow execution under three successively more realistic sets of assumptions:

(1) both the number of processing elements (PE's) and the fan-in and fan-out at each operator are unlimited (the "ideal" case),

(2) an unlimited number of PE's, but fan-in and fan-out is restricted to two,

(3) only 256 PE's and fan-in and fan-out is restricted to two.

(In the last case, the limitation of having only 256 processors matters only when the parameters of the problem are sufficiently large.)

	CONVENTIONAL		
	MVA	NCR	NCD
operations	$4NK$	$2NK$	$KL+\log_2 N$
space	K	$\min(N,K)$	K
	DATA FLOW		
	MVA	NCR	NCD
operators	$3NK$	$2NK$	$K[N+L]$
ideal	$5N$	$2[N+K]+2$	2
2-way fan	$N[3+\log_2 K]$	$2[N+K]+2$	$N+\log_2(KL)$
256 PE's	$\frac{3NK}{256}$ if $K>256$	$\frac{2NK}{256}$ if $\min(N,K)>256$	$\frac{K[N+L]}{256}$ if $L>256$

Table 1. Approximate Computational Costs in the Single Class Case.

Note that the details of these expressions for computational cost may vary somewhat depending on how one envisions the algorithm being implemented. For our purposes in this study, it is only important that the dominant terms be correct.

5. Discussion

5.1. Theoretical View

5.1.1. Single Class

On sequential architectures, exact solution of a single class queueing network model requires about $4NK$ operations. Assuming an unlimited number of processing elements, and an unlimited fan-in and fan-out at each one, the data flow architecture allows the solution time to be reduced to $O(N)$ by Mean Value Analysis, $O(K+N)$ by Normalization Constant Recursion, and to $O(1)$ by Normalization Constant Direct computation. With MVA, all devices can be treated in parallel, whereas with NCR there is a double recursion, leading to the K term in the computational complexity. The constant time of NCD comes only with fan-in and fan-out degrees greater than L, which is totally unrealistic for all networks of interest.

With the more realistic assumption that fan-in and fan-out at each operation is restricted to two, the relative costs of the three methods are quite different. The NCR approach is unaffected since the recursive calculation of the normalization constant is naturally done with operations having only two inputs and two destinations for the output. With MVA, on the other hand, the summation of residence times over all devices adds a time factor of $\log_2(K)$ at each population level. On conventional architectures, MVA and NCR are usually thought to have the same orders of computational cost, so the difference between NCR $(O(N+K))$ and MVA $(O(N\log_2 K))$ in the case of data flow architectures with limited fan-in and fan-out is interesting. For NCD, the products over K devices require $\log_2 K$ time steps, and the summation over all feasible states requires $\log_2 L$ time steps. Since the $\log_2 L$ term (which is essentially $\log_2(N!)$) grows much faster than N, the NCD method is not of interest for any models of realistic size (i.e., $N \geq 4$ and $K \geq 4$).

For sufficiently large models, the parallelism in all three solution approaches will become so high that all available processing elements might be kept busy continually for most of the computation. In this case, with a good compiler for the data flow language, the computation rate could come close to the product of the number of processing elements times the computation rate of each one.

5.1.2. Multiple Class Models

The multiple class results are almost direct generalizations of the single class results. In the "ideal" case, however, the NCR algorithm can exploit high degrees of fan-in to do all computations corresponding to a particular center in parallel, making it appear much superior to MVA. With fan-in degree restricted to two, NCR remains better than MVA (assuming that $\sum_c N_c$ exceeds K) by

avoiding the recursion over customer population levels. The NCD approach remains acceptable (and even preferable!) for slightly larger models than in the single class case. This is true because the more classes there are, the longer L stays small relative to $\prod_c (N_c+1)$. For all three approaches, very high potential parallelism is feasible already for quite small models, so again the number of processing elements will be the limitation on computation rate in most cases of interest (assuming that the compiler is very effective in balancing the computational load across all processing elements).

	CONVENTIONAL		
	MVA	NCR	NCD
operations	$4CK\prod_c(N_c+1)$	$CK\prod_c(N_c+1)$	$KL+K\sum_c \log_2 N_c$
space	$K\prod_c(N_c+1)$	$K\prod_c(N_c+1)$	$\sum_c KN_c$
	DATA FLOW		
	MVA	NCR	NCD
operators	$6CK\prod_c(N_c+1)$	$CK\prod_c(N_c+1)$	$K\left[LC+\sum_c N_c\right]$
ideal	$6\sum_c N_c$	$2K$	2
2-way fan	$2[\log_2 C+\log_2 K]\sum_c N_c$	$K[\log_2 C+\sum_c \log_2(N_c+1)]$	$\max_c N_c+\log_2(CKL)$
256 PE's	$\frac{6CK\prod_c(N_c+1)}{256}$ if $\sum_c N_c>8$	$\frac{CK\prod_c(N_c+1)}{256}$ if $\frac{CK\prod_c(N_c+1)}{N_{max}}>256$	$\frac{K\left[LC+\sum_c N_c\right]}{256}$ if $L>256$

Table 2. Approximate Computational Cost in the Multiple Class Case.

5.2. Practical View

In this section, we consider what kinds of queueing network models could conceivably be solved on data flow machines even though they are infeasible on conventional machines. We will consider conventional solutions to be acceptable if they require fewer than 10^7 operations (since that many operations can be executed sequentially in well under 10 seconds on current machines). We will consider conventional solutions to be infeasible if they require more than about 10^{10} operations (since this would require several hours of sequential computation on almost all current machines).

5.2.1. Exact Solutions

Because most single class models have parameters that satisfy

$$4NK \leqslant 10^7 ,$$

the use of data flow architecture would not significantly expand the class of useful single class queueing network models.

In the case of multiple class models, we must consider two cases distinguished by whether or not the model is sufficiently large that essentially all available processing elements will be busy or not. Examining the thresholds shown in Table 1 for when all processing elements become busy, it is apparent that whenever

$$4CK \prod_c (N_c + 1) > 10^7 \qquad (5)$$

all three of the dataflow approaches will lead to sufficient parallelism to keep as many as 256 processing elements busy essentially all of the time. Consequently, the computation time of a solution will be at least as great as the number of required operations divided by the product of the number of processing elements and the operation execution rate of each one. When the inequality above is not satisfied, the conventional solution is acceptable, and a data flow architecture does not increase the class of queueing networks that are feasible for solution.

A high degree of parallelism (say with 256 processing elements) along with very fast processors means that, for very large queueing network models, data flow architectures might make it feasible to solve networks requiring as much as 1000 times as many operations as are required for the solution of the largest networks solvable on conventional machines. Let us consider what this factor of 1000 means in terms of the variety of networks that can be solved.

The total number of operations depends on three parameters: the number of devices, the number of classes and the number of customers per class. Because the number of devices (K) is a multiplicative factor in the computational cost (see equation (5)), with 1000 times more operations, models with 1000 times more devices can be solved. However, models with many hundreds of devices are solved with conventional architectures, and limitations on number of devices generally come more from space restrictions than from time restrictions.

The factor by which the number of customers per class can be increased (assuming that each class has its population increased by the same factor)

clearly depends on the number of classes in the model. In general, the factor is $(1000)^{1/C}$. For single class models, this means that the population can be increased by a factor of 1000, while for a two-class model, the factor drops to about 32. For models of five and ten classes, the respective factors are 4 and 2. Thus, substantial gains in population size per class are possible only when the model has very few classes.

Finally, we consider the possible increase in the number of classes. We will assume that all classes have the same populations, so that the computational cost expression specializes to:

$$4CK[(N_c+1)^C]$$

Since the class population is raised to the power of the number of classes, the number of additional classes that can be added when increasing the amount of computation 1000-fold is very sensitive to the class population. If the class population is extremely small (say 1 or 2), then the number of classes added can be as high as seven to ten. With populations around 3 or 4, the number is four to six additional classes. When the class population is 10, only about two or three classes can be added, and when the class population exceeds 25, then at most one or two classes can be added to the model without expanding its computational requirement by more than a factor of 1000.

In summary, the use of data flow architecture machines would make it possible to solve somewhat larger models than can be solved practically on conventional machines, but the combinatorial nature of the computational cost expression means that a 1000-fold gain in computational power permits only much smaller changes in the number of classes in the model or the number of customers per class. Thus, from a practical point of view, it is not clear that the use of data flow architectures for solving queueing network models would have a significant impact on the class of queueing networks found to be useful.

5.2.2. Approximate Solutions

Throughout this paper, we have concentrated on the "exact" solution of queueing network models. However, for very large networks, the extraordinarily large number of computational steps may mean that a significant numerical error in the results will be a danger, so that the results can no longer be considered "exact". It is often the case that close "approximate" solutions to the queueing network model are acceptable since model input parameters are seldom known precisely.

One technique that has proven useful in reducing the computational requirements for analyzing very large models is "hierarchical modelling." Particularly if there is some symmetry in the large system, it may be possible to solve subsystems in isolation with low level models, and then to solve the interactions among the subsystems in a higher level model. Such an approach can greatly reduce the total number of computational steps required to solve the model.

Alternatively, the use of one of the approximate solution algorithms derived from mean value analysis [S79, CN82] may be more appropriate for treating very large models than use of the exact algorithms such as MVA or NCR. The

approximate MVA algorithms have computational costs that are independent of the class populations (such as $O(KC)$ or $O(KC^3)$). Consequently, in all practical cases, their application requires a number of operations that is small relative to 10^7, and they can be executed acceptably on conventional machine architectures.

6. Conclusions

For several reasons, it does not seem likely that the availability of highly parallel machines for queueing network analysis will significantly change the type or size of queueing networks used to model computer system and communication network performance. First, queueing network analysis algorithms can handle extremely large single class models quickly on sequential machines. Even with multiple class models, a thousand-fold parallel machine would only allow the solution of models with a few more classes and/or a few more customers per class than in models that are currently solved on sequential machines.

Second, because the goals of modelling include simplicity, understandability, and conciseness (with respect to the number of parameters), it may be the case that the sizes of models found useful will not increase even though the systems being modelled will be larger and parallel machines would be able to solve the larger models effectively.

Third, because it is seldom necessary to determine the exact solution of large models (due to uncertainties already present in the parameter values among other things), large queueing network models can often be solved either by a hierarchical modelling approach or by using an approximate solution technique derived from mean value analysis. Either of these approaches radically reduces the amount of computation required to solve large queueing network models.

An observation of interest in this study is the fact that parallel architectures lead to a better gain in performance for convolution based algorithms (NCR) than for those based on mean value analysis (MVA). In their sequential forms, convolution and mean value analysis algorithms are generally considered to have essentially equivalent computational cost.

Perhaps the most significant lesson to be learned from our study is the fact that starting from an algorithm statement with maximal apparent parallelism (i.e., minimal data dependency) does not always lead to the best implementation of the algorithm for a parallel machine. In our study, the NCD algorithm had the least data dependency. However, with the realistic assumption that fan-in and fan-out at each data flow component is limited to some small number, the large number of factors in each term and the vast number of terms to be summed cause the NCD approach to be much less desirable than either the NCR or MVA approach except for extremely small models. With limited fan-in and fan-out, the number of component stages required to distribute parameter values to where they are needed in the NCD scheme, and the number of stages required to sum over all terms become intolerably large when large models are analyzed. The NCR and MVA schemes use recursion (each in a different way) to avoid the need for distributing parameter values widely and calculating the sum of a very large numbers of terms. Thus, the computation path length of the data flow

machine is much shorter for NCR and MVA than for NCD.

7. References

[ABD85]

G. B. Adams, R. L. Brown, and P. J. Denning, Report on an Evaluation Study of Data Flow Computation, RIACS TR 85.2, Research Institute for Advanced Computer Science, NASA Ames Research Center (April, 1985).

[B73]

J. P. Buzen, Computational Algorithms For Closed Queueing Networks With Exponential Servers, *CACM 16*, 9 (September 1973), 527-31.

[CN82]

K. M. Chandy and D. Neuse, Linearizer: A Heuristic Algorithm For Queueing Network Models Of Computing Systems, *CACM 25*, 2 (February 1982), 126-33.

[D84]

J. B. Dennis, Data Flow Ideas For Supercomnputers, Proc. COMPCON '84 Conference (February, 1984), pp. 15-19.

[DB78]

P. J. Denning and J. P. Buzen, The Operational Analysis of Queueing Network Models, *Comp. Surv. 10*, 3 (September 1978), 225-61.

[LZGS84]

E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*, Prentice-Hall, Englewood Cliffs (1984).

[RK75]

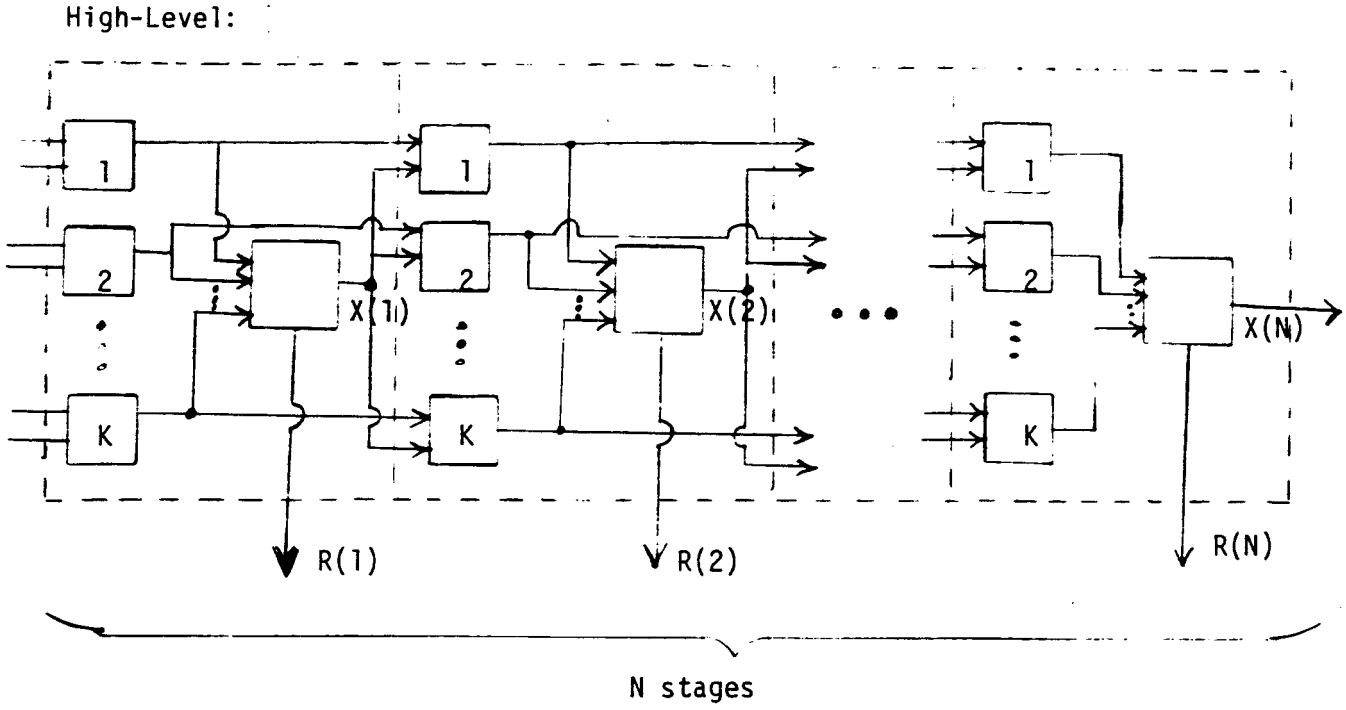
M. Reiser and H. Kobayashi, Queueing Networks With Multiple Closed Chains: Theory and Computational Algorithms, *IBM J. R&D 19*, (May 1975), 283-94.

[RL80]

M. Reiser and S. S. Lavenberg, Mean Value Analysis of Closed Multichain Queueuing Networks, *JACM 27*, 2 (April 1980), 313-22.

[S79]

P. J. Schweitzer, Approximate Analysis of Multiclass Closed Networks of Queues, *Proc. Int. Conf. on Stochastic Control and Optimization* (1979).



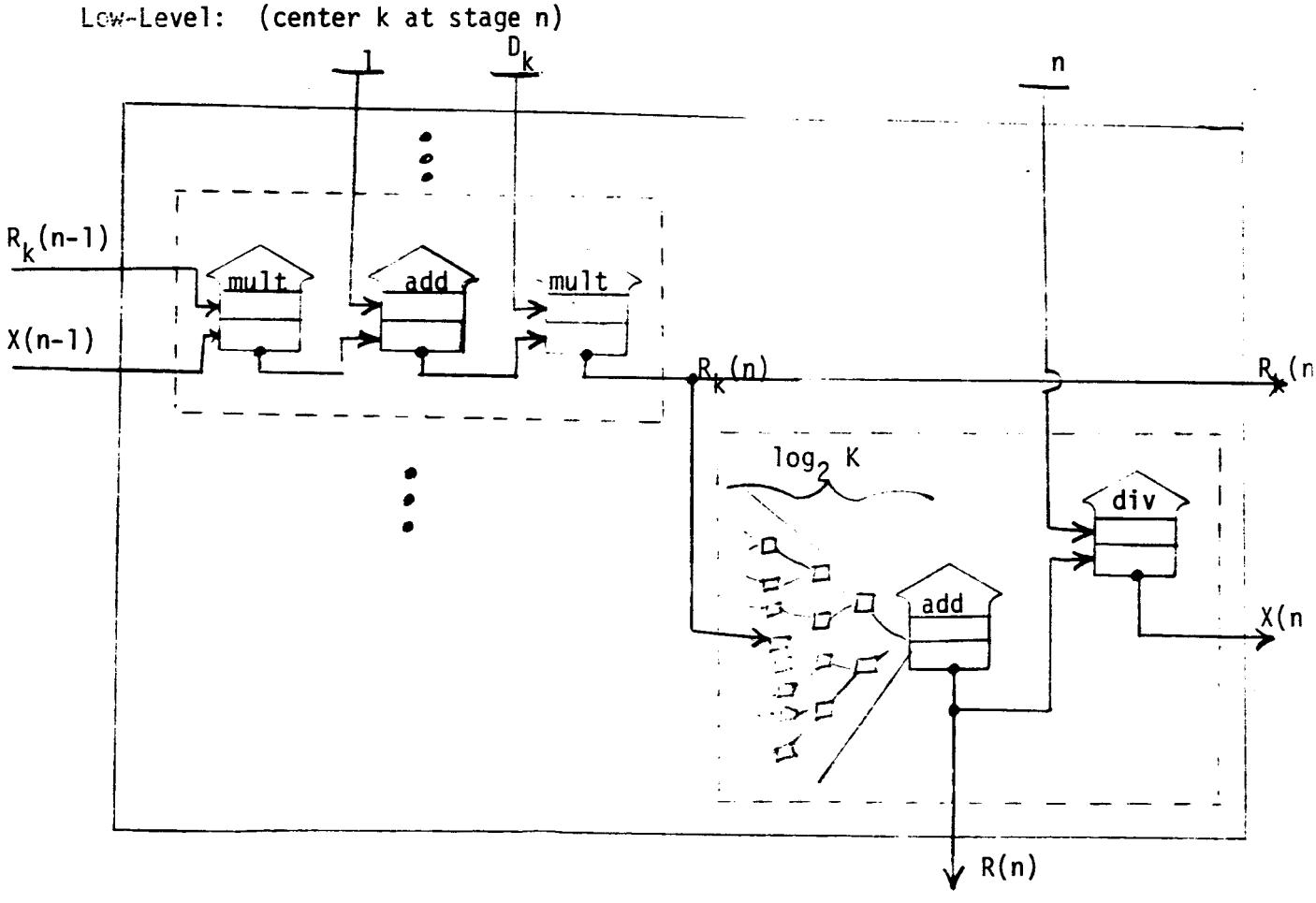


Figure 1(a). Data Flow Diagram For Single-Class MVA.

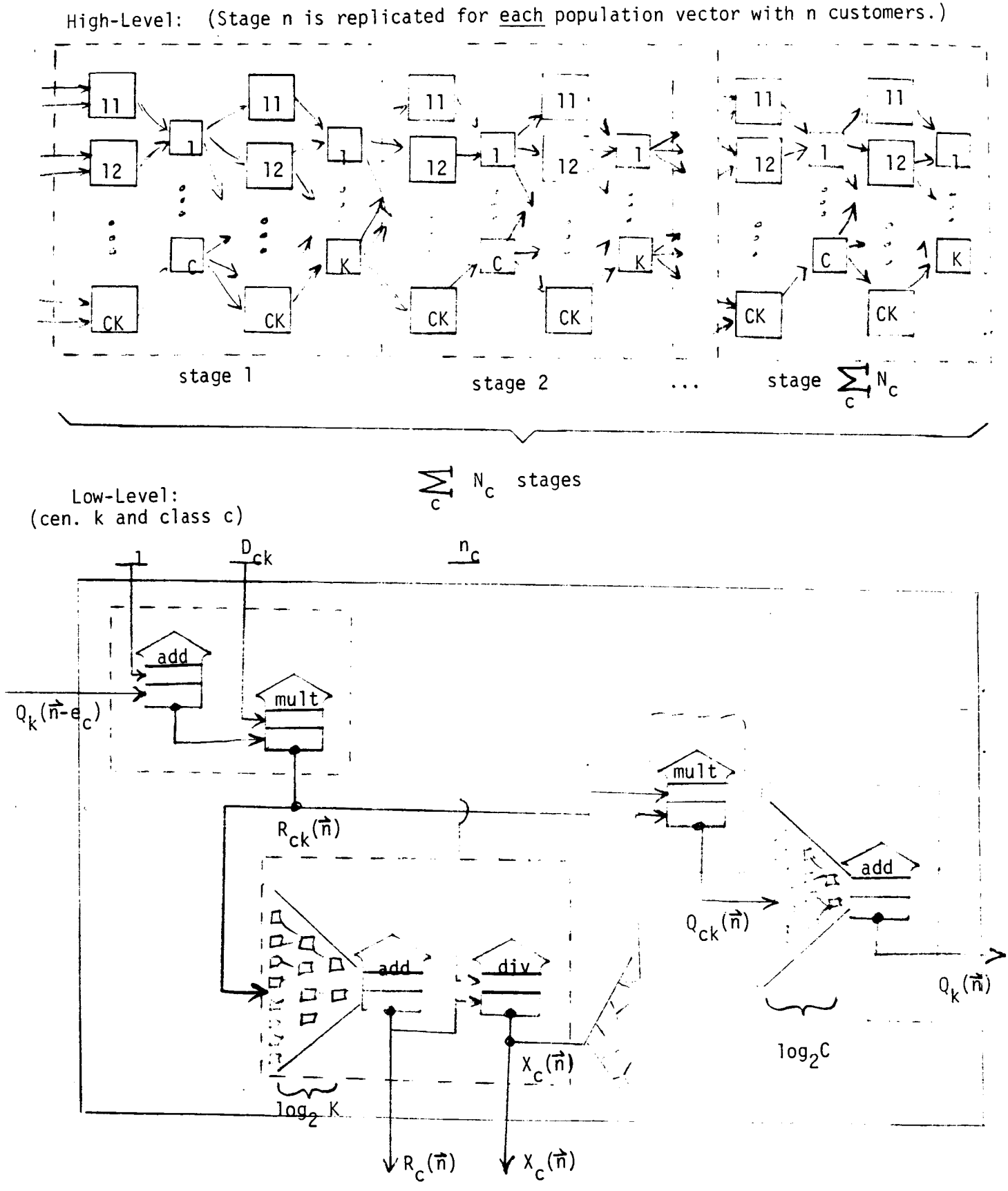


Figure 1(b). Data Flow Diagram For Multiple Class MVA.

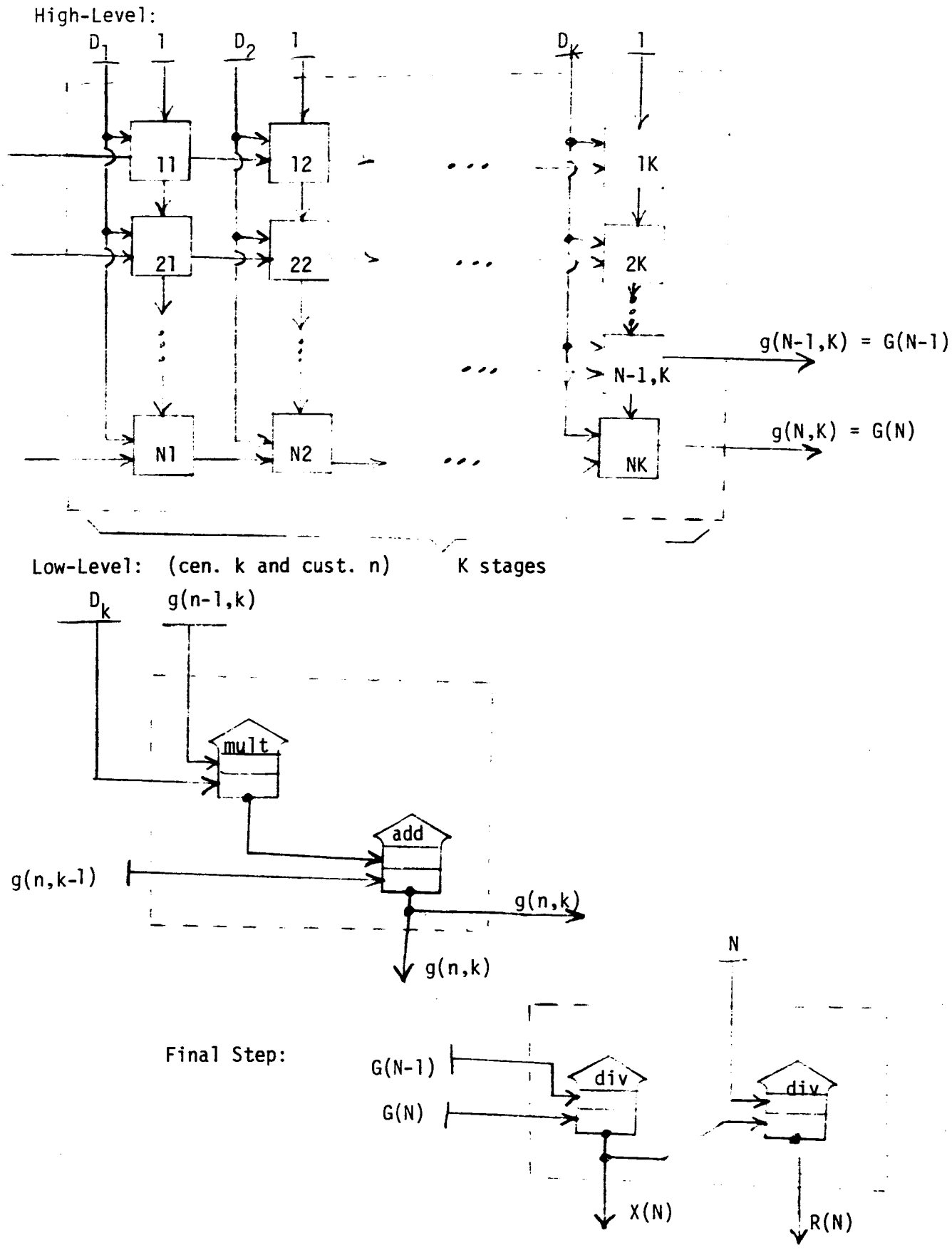


Figure 2(a). Data Flow Diagram For Single Class NCR.

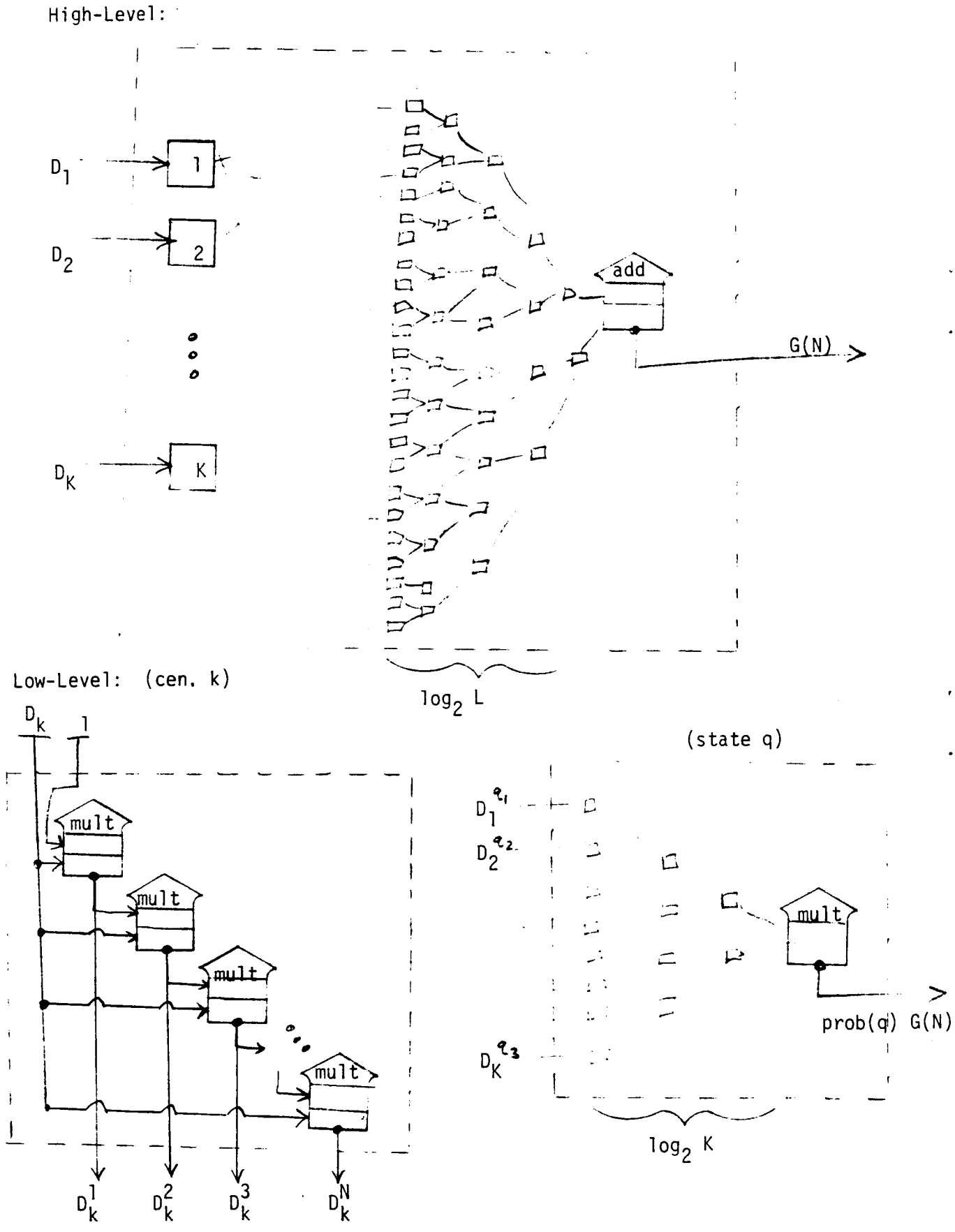


Figure 3(a). Data Flow Diagram For Single Class NCD.

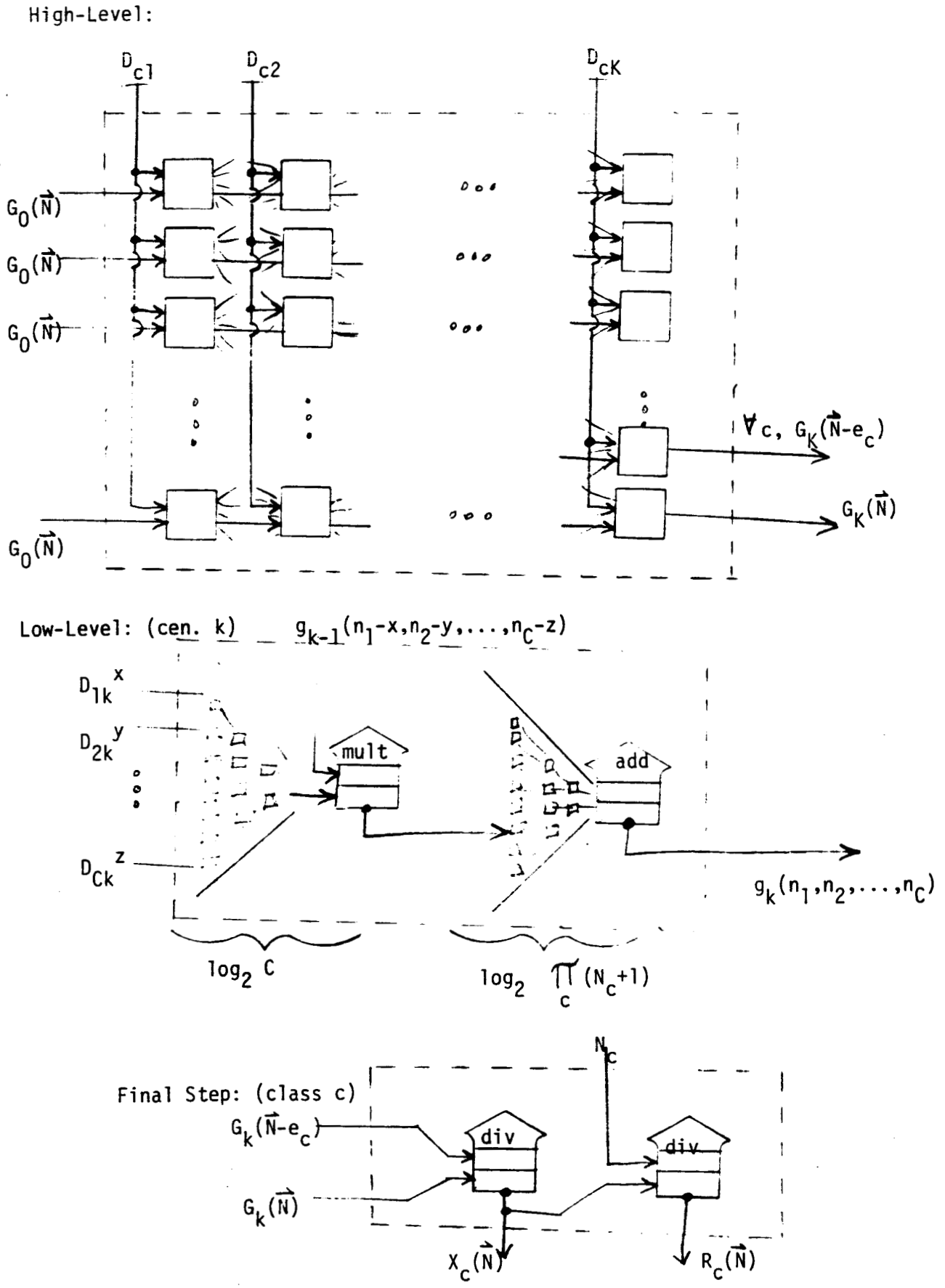


Figure 2(b). Data Flow Diagram For Multiple Class NCR.

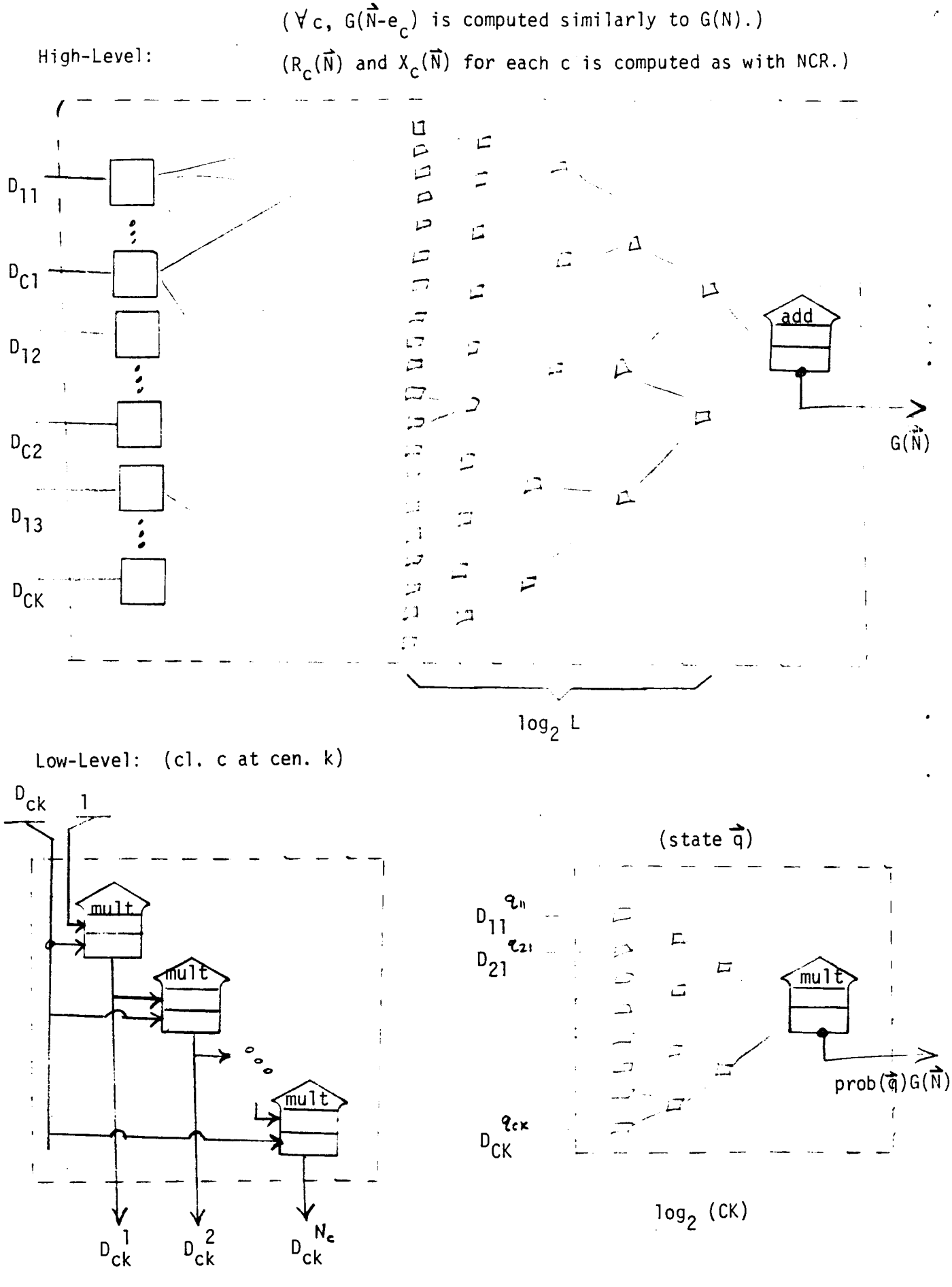


Figure 3(b). Data Flow Diagram For Multiple Class NCD.